Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jure Merčun

# Uvedba agilne metodologije razvoja na sistemu za vodenje podjetja

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn

Ljubljana 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Klasične metode razvoja programske opreme temeljijo na dobri analizi in načrtu celotnega projekta, ki morata biti v celoti zaključena pred začetkom programiranja. Na nekaterih projektih pa to ni mogoče, ker zahteve niso znane vnaprej, ali pa se že vnaprej pričakuje, da se bodo med razvojem spreminjale. V takem primeru je bolje uporabiti agilne metodolodije, ki namesto dobrega načrta temeljijo na sposobnosti sprotnega prilagajanja spremembam.

Študent naj na primeru konkretnega projekta enega od slovenskih podjetij, ki je pri vodenju obstoječega projekta prešlo s tradicionalne na agilno metodologijo, opiše nekaj agilnih metodologij, pri čemer naj največ pozornosti posveti izbrani metodologiji Scrum. Opiše naj razloge za prehod in analizira vpliv prehoda na projekt. Na koncu naj razišče možnost izboljšav v prihodnosti.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jure Merčun, z vpisno številko **63020106**, sem avtor diplomskega dela z naslovom:

*Uvedba agilne metodologije razvoja na sistemu za vodenje podjetja*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luke Šajna,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 18. septembra 2014 Podpis avtorja:

*Zahvaljujem se mentorju, doc. dr. Luki Šajnu za strokovno vodstvo*
*prof. dr. Francu Solini za nasvete in pomoč,*
*Nataši za moralno podporo*
*ter mami Zdenki in očetu Andreju za desetletje spodbude.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **ERP** | enterprise resource planning | načrtovanje virov podjetja |
| **QA** | quality assurance | nadzor kakovosti (testiranje) |
| **XP** | extreme programming | ekstremno programiranje |
| **DoD** | definition of done | definicija končane zahteve |
| **WIP** | work in process | delo v teku |
| **TDD** | test driven development | testno usmerjen razvoj |

# Povzetek

Diplomska naloga temelji na konkretnem primeru prehoda s tradicionalnih na agilne metodologije razvoja programske opreme na projektu s področja načrtovanja virov podjetja, ki poteka v enem od slovenskih podjetij. Opisuje težave, s katerimi se je razvojna ekipa spopadala pri tradicionalni metodi dela, in razloge za prehod na agilne metodologije. Opisane so tri agilne metodologije, na kratko ekstremno programiranje in kanban, ter bolj podrobno Scrum, ki je bil na koncu izbran za vodenje projekta. Poleg metodologije so opisane tudi prilagoditve, ki so bile izvedene na projektu, kjer smo zaradi specifik projekta ocenili, da osnovna metodologija ni ustrezna. Sledi analiza, v kolikšni meri so bile v uvodu opisane težave s tem odpravljene. Na koncu so opisane še morebitne izboljšave v prihodnosti.

**Ključne besede:** projektno vodenje, razvoj programske opreme, agilne metodologije, scrum.

# Abstract

This thesis is based on an actual case of transitioning from a traditional to an agile software development methodology on an enterprise resource planning project, which is being developed in one of Slovenian companies. It describes the problems, that the development team faced with the traditional method and the reasons for transitioning to an agile methodology. It includes brief descriptions of extreme programming and kanban, and a more in depth description of Scrum, which was the chosen methodology for the project. It also describes certain customizations of the method, which were deemed necessary due to specific nature of the project, followed by an analysis of the original problems and how they were affected by the change in methodology. In the end, possible improvements for the future are addressed.

**Keywords:** project management, software development, agile software development, scrum.

# Poglavje 1

# Uvod

Razvoj programske opreme je proces načrtovanja, programiranja, testiranja in vzdrževanja programske opreme. Ta opis je sicer ustrezen, nam pa zaradi raznolikosti programske opreme ne pove veliko. Po eni strani gre lahko s stališča projektnega vodenja za zelo preprost projekt, ki zahteva vrhunsko znanje z nekega specifičnega področja. Po drugi strani gre lahko s programerskega stališča relativno preprost projekt, ki pa zahteva mnogo vsebinskega znanja in sodelovanje večjega števila programerjev, svetovalcev in drugih sodelujočih. Če je v prvem primeru projektno vodenje bolj ali manj trivialno, pa lahko v drugem primeru napačen pristop zelo oteži napredek oz. ga popolnoma ustavi. Zato je potrebno izbrati pristop, ki ustreza okoliščinam.

Pričujoče diplomsko delo opisuje konkreten projekt, ki je bil najprej voden na klasičen način, vendar se je le-ta izkazal za neučinkovitega. Opisuje glavne težave, s katerim se je spopadala ekipa in razloge za prehod na agilen način razvoja. Sledi opis nekaterih agilnih metodologij in opis prehoda na *Scrum* ter prilagoditve same metodologije, ki so bile potrebne zaradi specifičnih lastnosti projekta. Sledi analiza, v kolikšni meri je prehod na agilne tehnologije izboljšal proces in rešil prej opisane težave. Zaključek vsebuje še nekaj predlogov za izboljšave v prihodnosti.

## 1.1 Projekt

Pojekt obsega implementacijo ERP rešitve za konkretnega naročnika na slovenskem trgu. Pokriva področja računovodstva, poslovanja s kupci in dobavitelji,

kadrovske evidence in pogodbenih razmerij. Najpomembnejši del implementacije pa je področje zaračunavanja storitev kupcem, ki je tudi najobsežnejši. Poleg razvoja in implementacije rešitev je ekipa zadolžena tudi za podporo uporabnikom.

### 1.1.1 Ekipa

Ekipa je bila v osnovi sestavljena iz desetih članov.

#### Projektni vodja

Projektni vodja skrbi predvsem za operativno vodenje. Pazi, da se zahteve rešujejo, do neke mere skupaj s stranko določa prioritete, preverja porabljen čas, ugotavlja zaračunljivost ur in pazi, da vse poteka skladno s pogodbami.

#### Arhitekt

Arhitekt skupaj z vodjo razvoja skrbi za tehnične odločitve glede arhitekture ter skupaj z vodjo svetovalcev išče vsebinske rešitve.

#### Razvijalci

Razvijalci so zadolženi za razvoj (programiranje) zahtevkov, grobo testiranje ter iskanje in popravljanje napak v obstoječi kodi. V pomoč uporabnikom praviloma niso direktno vpleteni, razen kadar je potrebna masovna manipulacija s podatki, ki je uporabniki ali svetovalci ne morejo izvesti, ali pa bi ročno trajala predolgo. V ekipi je pet razvijalcev.

#### Svetovalci

Naloga svetovalcev je komunikacija s strankami in podpora pri delu. Analizirajo zahteve strank in generirajo zahtevke. V primeru prijave napak mora svetovalec ugotoviti, ali gre za napako s strani uporabnika, ali gre za napačno delovanje sistema, ki ga mora popraviti razvijalec. Tekom razvoja svetovalci skrbijo za testiranje kode. V ekipi so tri svetovalke, ki ne programirajo, saj so zadolžene za vsebino. Zato so po izobrazbi z ekonomskega področja, saj velik del vsebine obsega računovodstvo in finance.

### 1.1.2 Razvojno okolje in orodja

Razvojno okolje projeka je ERP rešitev Microsoft Dynamics AX 2012 (krajše *AX*). AX je sistem za vodenje večjih podjetij, z že implementiranimi rešitvami za večino področij poslovanja podjetij – glavna knjiga, saldakonti kupcev in dobaviteljev, proizvodnja, javna naročila, kadrovska evidenca, itd. – ki pa ne pokriva prilagoditev za slovenski trg. Vsebuje pa vso potrebno izvorno kodo poslovne logike in omogoča poljubne prilagoditve.

Za vodenje projekta se uporablja orodje Atlassian JIRA. Omogoča beleženje in upravljanje z zahtevki, ki so razdeljeni v projekte in sledijo določenim delovnim tokovom. Omogoča visoko stopnjo prilagodljivosti, tako kar se tiče dodajanja novih podatkovnih polj na zahtevke, kot tudi spreminjanja delovnih tokov, tako da ustrezajo načinu razvoja. Omogoča tudi razvoj lastnih razširitev, ki omogočajo dodatne prilagoditve. JIRA med drugim omogoča beleženje porabljenega časa, za kar je bila npr. razvita integracija z internim sistemom za izdajo faktur strankam. Za podporo agilnim procesom vsebuje razširitev *Greenhopper*.

# Poglavje 2

# Klasični pristop

To poglavje opisuje klasični kaskadni model in obliko, ki je bila uporabljena na projektu, ter težave, ki so se pri tem pojavljale.

## 2.1 Kaskadni model

Kaskadni model (angl. *Waterfall*) predvideva pet faz razvoja[1]:

1. Analiza
2. Načrtovanje
3. Kodiranje
4. Testiranje
5. Vzdrževanje

Faze si sledijo po vrsti. Določena faza se lahko začne šele, ko je predhodnja faza zaključena, na prejšnje se ne vrača. Najprej naredimo analizo celotnega projekta. Od naročnika dobimo vse zahteve, ugotovimo kakšen je obseg projekta, dokumentiramo procese. Ko zberemo vse potrebne podatke, naredimo načrt poteka dela. Sledi programiranje zahtevanih funkcionalnosti in testiranje. Ko je projekt predan naročniku sledi še vzdrževanje in podpora.

Glavna pomanjkljivost kaskadnega modela je njegova rigidnost. Zaradi sekvenčne narave modela morajo biti vse zahteve znane vnaprej. Model ne predvideva vračanja nazaj na analizo, če se zahteve spremenijo, ko je razvoj že v teku.

V primeru, da pride do sprememb, sta ponovno potrebna analiza in načrtovanje. Spremembe praviloma niso izolirane in povzročijo, da del analize in načtrovanja postane zastarel in ga je potrebno ponoviti, kar seveda pomeni izgubo časa.

## 2.2 Glavne težave

Med razvojem se je ekipa spopadala z različnimi problemi, od katerih se jih je nekaj redno ponavljalo. V tem poglavju so opisani bistveni problemi, ki so oteževali delo in so bili tudi glavni razlogi za iskanje nove metodologije. Podobni problemi so očitno doleteli tudi druge ekipe [2].

### 2.2.1 Spreminjanje prioritet

Tekom razvoja so se prioritete pogosto spreminjale. V nekaterih primerih je šlo za zunanje faktorje – spremenjene razmere na trgu, ali delo z zunanimi sistemi – v drugih primerih pa bodisi za spremembe usmeritve projekta na drugo področje, bodisi za usklajevanje dela za različne oddelke.

### 2.2.2 Spreminjanje in dodajanje zahtev

Druga težava je bila spreminjanje zahtev. Sčasoma se je izkazalo, da prvotni načrt ne ustreza potrebam. Nekatere zahteve so se spremenile zato, ker se je v času načrtovanja vedelo, da so potrebne določene funkcionalnosti, ne pa tudi na kakšen način bodo implementirane. Spet druge zahteve so se pojavile povsem na novo, ker se med načrtovanjem v resnici ni vedelo zanje. Pogosto so bile – oz. so se zdele – te zahteve dovolj nujne, da se je od razvojne ekipe pričakovalo takojšnji odziv.

### 2.2.3 Slabo testiranje

V začetni fazi projekta je vladal konstanten pritisk za razvoj novih funkcionalnosti, ki jih je bilo potrebno implementirati do zastavljenih rokov. Obenem je obstjala na strani naročnika množica uporabnikov, ki še ni bila pripravljena na samostojno delo, zaradi česar je svetovalkam zmanjkovalo časa za testiranje. Posledično je prišlo v produkcijsko okolje precejšnje število napak, katerih odpravljanje je povzročalo dodatno izgubo časa.

### 2.2.4 Slaba definicija

Pogosto so od naročnika prišle zahteve, ki enostavno niso bile dovolj dobro definirane, da bi jih razvojna ekipa lahko rešila. Jasna definicija je pogosto prišla tik pred rokom, ko bi morala biti rešitev že implementirana, ali pa se je ekipa lotila nove zahteve, ki sicer ni bila popolnoma definirana, a je bila dovolj nujna, da je bilo potrebno začeti. Ideja je bila, da se med samim razvojem ključne funkcionalnosti od naročnika izvejo še manjkajoče potrobnosti, vendar se je pogosto zgodilo, da se je vmes našlo nekaj bolj nujnega, zaradi česar so ti detajli ostajali nerešeni, zahteva pa le na pol implementirana.

### 2.2.5 Naraščanje obsega

Prvotne zahteve so sčasoma spet postale aktualne in jih je bilo treba rešiti do konca, vendar pa se je pri tem ponavadi pojavila kakšna dodatna podrobnost, ki je v izvirni zahtevi ni bilo, vendar je uporabnikom prišla na misel med uporabo. To je povzročilo povečan obseg projekta (*Scope creep*), zaradi česar razvoj ni več dohajal rokov (kot je razvidno iz prejšnjega odstavka, jih v resnici ni dohajal niti naročnik s podajanjem zahtev), ki so kljub naraščanju obsega ostali v prvotnih okvirih. To je privedlo do nadurnega dela in nekajkrat tudi do delovnih sobot, kar je negativno vplivalo na produktivnost [3].

## 2.3 Iskanje boljše metodologije

Sčasoma se je pokazalo, da ni šlo za časovno omejen pojav, ampak je bilo očitno, da se bodo na tak način težave nadaljevale na dolgi rok. Ker omenjeni projekt ni bil edini tak primer – s podobnimi težavami so se spopadali praktično vsi ERP projekti v podjetju – se je vodstvo podjetja odločilo za najem zunanjega strokovnjaka na področju optimizacije delovnih procesov v IT. Ankete in osebni razgovori z vsemi zaposlenimi, ki smo delali na teh projektih, so pokazali, da smo v veliki večini zadovoljni s podjetjem, delom in delovnim okoljem.

Po drugi strani pa smo praktično vsi zaposleni negativno ocenili kvaliteto analize zahtev, realnost in transparentnost časovnega načrta in rokov, jasnost metodologije dela in razpoložljivost namenske ekipe za zagotavljanje kakovosti. Potrebno

je bilo najti bolj primerno metodologijo, s katero bo mogoče slediti hitrim spremembam zahtev.

# Poglavje 3

# Agilne metodologije

Agilne metodologije temeljijo na kratkih ciklih načrtovanja in razvoja in pogostih (ter posledično majhnih) izdajah. Spreminjanje zahtev se v agilnih metodologijah pričakuje [4], zato se tudi spodbuja postopno definiranje zahtev — zahteve se dokončno definirajo šele tik pred razvojem, saj se ravno zaradi pričakovanih sprememb prezgodnje načrtovanje lahko izkaže za izgubo časa.

Agilne metode spodbujajo delo v relativno majhnih, samoorganiziranih ekipah. Poudarja se samostojno iskanje razvojnih rešitev, s čimer se poveča motivacija razvojne ekipe. Prav tako se spodbuja dnevna komunikacija med razvojno ekipo in poslovnimi uporabniki. Zaradi kratkih iteracij se od razvojne ekipe pričakuje konstantna produktivnost, zato je nadurno delo nezaželjeno.

Ob rednih intervalih (navadno enkrat na iteracijo) si ekipa vzame čas za refleksijo o preteklem delu in možnosti za izboljšave delovnega procesa. Konstruktivne predloge nato vključi v naslednje iteracije.

Uspešnost agilnih metod potrjujejo tako svetovne, kot domače raziskave [5, 6]. V nadaljevanju sledi kratek opis dveh znanih agilnih metodologij in daljši opis metodologije *Scrum*, ki je bila izbrana za nadaljni razvoj projekta.

## 3.1 Ekstremno programiranje

Ekstremno programiranje [7] je agilna metodologija, ki je ime dobila po svojem pristopu – če je praksa dobra, potem se jo pelje do ekstrema, npr.: pregled kode je koristen, zato naj se kodo pregleduje ves čas. XP zato zagovarja programiranje

v paru.

Osnovni principi XP so:

- Hiter odziv.
  Hitre povratne informacije so v XP najboljši način učenja, pa naj gre za odziv naročnika ali pa rezultate sprotnega testiranja.

- Predpostavljena preprostost.
  Vse zahteve se rešuje na najpreprostejši možen način, ki še ustreza zahtevam. S tem se prihrani dovolj časa, da se kasneje implementira dodatno kompleksnost za tiste zahteve, za katere se pojavi potreba (avtor ocenjuje, da je takih zahtev 2%). Pri programiranju se rešitve ne prilagaja glede na pričakovane nadgranje v prihodnosti.

- Stopnjujoče spremembe.
  Vse spremembe so majhne in postopne, saj velike spremembe ne funkcionirajo. Ekipa se spremembam ne sme upirati, ampak jih mora sprejeti.

XP prakticira programiranje v paru (*Pair programming*), kar pomeni, da dva razvijalca naenkrat uporabljata isto delovno postajo. Na ta način eden programira in se koncentrira na kodo, ki jo piše, medtem ko drugi sproti preverja kodo in razmišlja bolj na široko. Razvijalca pogosto menjata vlogi in tudi pari se med seboj pogosto menjajo.

Za zagotavljanje enotskih testov (*Unit tests*) se uporablja testno usmerjen razvoj (*Test driven development*). TDD predvideva, da razvijalec *najprej* napiše test – ki očitno pade – šele nato napiše kodo, ki opravi test. Napisana koda ne sme biti bolj kompleksna, kolikor je nujno, da opravi test. Če je potrebna dodatna kompleksnost, je treba najprej napisati nov test.

Razvijalci praviloma delajo na lokalni kopiji projekta. Ker se pričakuje, da ekipa dela vedno na najnovejši različici, se dnevno pričakuje večkratna oddaja sprememb v repozitorij, kar omogoča hitrejše odkrivanje problemov z integracijo.

## 3.2 Kanban

Kanban [8] je agilna metodologija, ki od ostalih izstopa z obratnim pristopom in se odraža v štirih osnovnih načelih:

- Začni z obstoječim procesom.
  Kanban ne predvideva drastičnih sprememb takoj na začetku, namenjen je upravljanju s spremembami.

- Strinjaj se z majhnimi, stopnjujočimi se sprembami.
  Optimizacija procesa poteka v majhnih korakih, saj velike in hitre spremembe povzročijo strah in odpor.

- Spoštuj trenutne procese, vloge in dolžnosti.
  Kanban skuša obdržati dele procesa, ki funkcionirajo. S tem tudi poskuša preprečiti strah pred spremembami.

- Vodenje na vseh nivojih.
  Spodbuja vodenje na vseh nivojih, od posameznikov do uprave.

Optimizacija sistema s kanbanom temelji na treh praksah. **Vizualizacija** na najbolj osnovnem nivoju pomeni, da se pripravi tablo (podobno, kot pri ostalih agilnih metodologijah), razdeljeno na stolpce, ki ustrezajo korakom v delovnem procesu (analiza, razvoj, testiranje, ...). Na ta način ekipa (in ostali zainteresirani) vedno vidijo stanje projekta.

Delo v teku (*Work in process*) pomeni število zahtev, ki se istočasno rešujejo. Z **omejevanjem dela v teku** dosežemo, da zahteve ne ostajajo odprte in so posledično hitreje rešene. Več kot je odprtih zahtev, daljši je potrebni čas (*Lead time*), v katerem je zahteva v procesu. Omejitev povzroči, da je v sistemu manj zahtev, ki čakajo. V primeru prestroge omejitve pa pade izkoriščenost ekipe, saj se pojavi preveč čakanja. Posamezne faze pa nimajo nujno enake meje, saj imajo lahko različne faze na voljo različno število ljudi. Pravilne vrednosti ekipa išče empirično. Kanban tudi dopušča, da se omejitev občasno prekrši, če zato obstajajo dobri razlogi.

Omejevanje WIP povzroči napetosti v delovnem toku, kar nam razkrije slabosti. **Upravljanje s tokom** pomeni iskanje ozkega grla, kjer se zahteve ustavljajo in način, kako ga odpraviti. Ko je le-to odpravljeno, se nam razkrije naslednje ozko grlo, ki ga rešujemo v naslednji iteraciji. Ta proces se v resnici nikoli ne konča, saj vedno ostane možnost za optimizacijo.

# Poglavje 4

# Scrum

Osnovna ideja Scruma je dejstvo, da je v praksi težko vse zahteve predvideti vnaprej. Zato je metodologija prilagojena hitri implementaciji majhnih kosov funkcionalnosti in prilagodljivosti spremembam.

## 4.1 Osnovni pojmi

### 4.1.1 Sprint

Sprint predstavlja eno iteracijo delovnega procesa in tipično traja dva to štiri tedne. Optimalno vsi sprinti trajajo enako dolgo, v nobenem primeru pa se ne spreminja trajanja že začetega sprinta [9]. Na začetku vsakega sprinta se določi cilje in merila, kdaj je sprint uspešno končan. Tekom sprinta se ne uveljavlja sprememb, ki bi ogrozile sprejete cilje ali zmanjšale kvaliteto implementiranih rešitev. Po potrebi se lahko med produktnim vodjo in razvojno ekipo izpogaja spremenjen obseg sprinta, vkolikor se tekom sprinta pojavi potreba po tem, vendar se dodatne zahteve dodajo za ceno opustitve časovno enakovrednega dela prvotnega načrta.

### 4.1.2 Seznam zahtev

Seznam zahtev (*Product backlog*) je seznam uporabniških zgodb (*User story*), ki jih naročnik potrebuje in je edini vir razvojnih zahtev. Seznam je urejen po prioriteti, tako da so najbolj nujne zgodbe na vrhu. Vrstni red zahtev na seznamu

je enak vrstnemu redu sprejema zahtev v sprinte. Projekt se ne začne s popolnim seznamom zahtev, ki se tekom projekta ena za drugo rešujejo, pač pa je na začetku projekta na seznamu le omejen nabor osnovnih zahtev, ostale pa se dodajajo kasneje. Seznam nikoli ni zares končan saj se s širjenjem funkcionalnosti in povratnimi informacijami uporabnikov vedno znova pojavljajo nove zahteve, ali pa se dodelujejo oz. na novo ovrednotijo stare zahteve. Zahteve na vrhu seznama morajo biti definirane do te mere, da jih je mogoče zajeti v sprint. Zahteve na dnu seznama pa tipično vsebujejo le grob opis. Za zahteve, ki bodo še dlje časa čakale na implementacijo, obstaja namreč precejšnja možnost, da se do implementacije še spremenijo. Zato se s podrobnim načrtovanjem čaka kolikor dolgo se da. Zgodbam na seznamu se določi grobe ocene zahtevnosti v točkah (*Story points*) [10]. Točka je abstraktna enota, ki ji vrednost ekipa določa po občutku. Koliko ur ali dni dela prestavlja točka v resnici ni pomembno, bistveno je, da cela ekipa operira z isto vrednostjo, kar omogoča hitro ocenjevanje zahtev brez natančnih časovnih ocen. S količino točk se tudi določi kapaciteta ene iteracije.

### 4.1.3 Definicija končane zahteve

Eden pomembnejših konceptov, ki ga je potrebno doreči pred začetkom prvega sprinta, je definicija končane zahteve (*Definition of done*). Samo zahteve, ki ustrezajo definiciji se namreč smatrajo za končane in samo sprint v katerem so vse zahteve končane se smatra za uspešno zaključenega. Namen definicije je, da se izogne sicer precej pogosti situaciji, v kateri se delno implemenirane ali slabo testirane spremembe vseeno prenesejo v produkcijsko okolje.

### 4.1.4 Sestanek za načrtovanje iteracije

Vsak sprint se začne s sestankom za načrtovanje iteracije (*Sprint planning*). Razvojna ekipa z vrha seznama zahtev vzame toliko zgodb, kolikor jih lahko reši v enem sprintu. S tem se ekipa tudi zaveže in prevzame odgovornost (*commitment*), da bo te zgodbe dejansko rešila. Pomembno je, da je ta seznam realen in rajši malce konzervativen [10]. Vkolikor na koncu sprinta ostaja čas, se lahko vzame dodatne zgodbe, medtem ko nezaključena zgodba lahko na koncu ogrozi integriteto celotnega sprinta. Na sestanku razvojna ekipa tudi doreče, na kakšen način bodo

zgodbe rešene. Zgodbe se razdeli na manjše naloge, ki jih lahko ekipa po potebi rešuje istočasno.

### 4.1.5 Dnevni sestanek

Vsak dan ob istem času se organizira 15-minutni dnevni sestanek (*Daily scrum* oz. *Daily standup*), na katerem člani razvojne ekipe poročajo o delu. Vsak član ekipe odgovori na tri vprašanja: [9]

- Kaj sem naredil od prejšnjega sestanka?
- Kaj bom naredil do naslednjega sestanka?
- Ali imam pri delu kakšne težave?

Prvi dve vprašanji sta namenjeni sprotnemu poročanju o napredovanju sprinta. Zadnje vprašanje je namenjeno sprotnemu odkrivanju ovir, ki bi utegnile zavleči napredek in ogroziti sprint. Pravočasno odkritje težav omogoča, da se razvijalcu še pravočasno nudi pomoč, da njegovo delo napreduje. Naloga skrbnika metodologije je, da poskrbi za redno izvajanje sestanka, samo izvedbo pa prevzamejo člani razvojne ekipe. Skrbnik metodologije prav tako pazi, da sestanek ne preseže predvidenega 15-minutnega okvirja. V ta namen se praviloma sestanek izvaja v stoje, da bi se kar najhitreje zaključil.

### 4.1.6 Sestanek za definiranje zahtev

Sestanek za definiranje zahtev (*Grooming*) je namenjen pripravi zahtev na vrhu seznama, ki bodo postale del enega od naslednjih sprintov. To lahko pomeni urejanje prioritet, pripravo podrobnejše razlage, razbijanje na več manjših, bolj obvladljivih zahtev ali razjasnitev kriterijev za uspešno končanje zahteve.

### 4.1.7 Sestanek za pregled rezultatov

Na koncu iteracije skrbnik metodologije organizira sestanek za pregled rezultatov (*Sprint review*), na katerem ekipa produknemu vodji in interesnim skupinam predstavi, kaj je bilo narejeno tekom sprinta. Sestanek je neformalne narave in

je namenjen predvsem zbiranju povratnih informacij, s pomočjo katerih se določi najverjetnejši obseg naslednje iteracije.

### 4.1.8 Retrospektiva

Retrospektiva je namenjena analizi iteracije in iskanju potencialnih izboljšav. Razvojna ekipa skupaj s skrbnikom metodologije predebatira dobre in slabe strani preteklega sprinta in ugotovi, kako bi se dalo delo izboljšati. Člani ekipe imajo možnost predlagati lastne ideje. Za vizualizacijo predlogov se lahko uporablja metode, kot je npr. morska zvezda (*Starfish*) [11] - na tablo narišemo petkrako zvezdo in dele označimo z:

*Začnimo . . . / Več . . . / Nadaljujmo z . . . / Manj . . . / Nehajmo . . .*

Potrebne spremembe napišemo v pripadajoča polja. Če se sprememb držimo, jih na naslednjih retrospektivah pomikamo v smeri *Začnimo* → *Več* → *Nadaljujmo* oz. *Manj* → *Nehajmo*. S table jih lahko odstranimo, ko postanejo del normalnega načina dela.

### 4.1.9 Scrum tabla

Scrum tabla (*Scrum board*) je tabla, na kateri je vedno razvidno stanje zgodb v iteraciji. Tabla je razdeljena na vrstice, ki predstavljajo vsaka svojo zgodbo, in stolpce, ki predstavljajo stanje zgodbe in njenih nalog. Označujejo se s samolepilnimi lističi. V prvem stolpcu (*Story*) je listič s kratkim opisom zgodbe, ki ostane na tem mestu celotno iteracijo. Ostali stolpci (*To do*, *In process*, *To verify*, *Done*) so namenjeni lističem z nalogami, ki pa se tekom iteracije premikajo proti desni glede na njihovo stanje.

## 4.2 Ključne vloge

### 4.2.1 Produktni vodja

Produktni vodja (*Product Owner*) je predstavnik naročnika. Njegova primarna naloga je skrb za seznam zahtev [9]. Produktni vodja skrbi za to, da seznam zahtev odraža potrebe naročnika. Pripravlja uporabniške zgodbe, ki jih umešča

na seznam zahtev, pri čemer pazi, da vrstni red sovpada s cilji in temu primerno pripravlja okvirni načrt iteracij. Zagotavlja, da je seznam zahtev jasno opredeljen in da odraža delo, ki ga bo razvojna ekipa opravljala v prihodnosti. Razvojni ekipi je na voljo z vsemi potrebnimi informacijami, ki jih potrebuje za uspešno izvedbo dela. Produktni vodja je vedno samo ena oseba, čeprav lahko predstavlja interese večih oseb ali entitet. Kakršnakoli sprememba seznama zahtev gre vedno preko produktnega vodje. Da je lahko njegovo delo uspešno, mora celotna organizacija spoštovati njegove odločitve. Nihče drug nima pravice razvojni ekipi določati prioritet, niti ne sme razvojna ekipa upoštevati navodil kogarkoli drugega. Produktni vodja lahko seznam zahtev vzdržuje sam oz. delo preda razvojni ekipi, vendar odgovornost ostaja izključno njegova. Vloga je ključnega pomena za uspešnost projekta, zato mora imeti produktni vodja pregled nad usmeritvijo in cilji celotnega projekta [5]. Produktni vodja je lahko del razvojne ekipe, zaradi narave njegovih nalog pa ne more biti obenem tudi skrbnik metodologije.

### 4.2.2 Razvojna ekipa

Razvojna ekipa (*Development team*) je zadolžena za implementacijo zahtev. Pokrivati mora vsa znanja, ki so potrebna za uspešen razvoj zahtevanih funkcionalnosti. Razvojna ekipa v vsaki iteraciji z vrha seznama zahtev vzame določeno količino zahtev, ki jih je zmožna v dogovorjenem času implemenirati in se tudi zaveže, da bodo ob koncu iteracije te zgodbe končane. Ker ekipa prevzame odgovornost za dokončanje iteracije ima tudi pravico do samoorganizacije - ekipa sama odloča, na kakšen način bo cilj dosegla. Nihče jim tega načina ne sme vsiljevati (niti skrbnik metodologije). Razvojna ekipa ne sme biti manjša od treh in večja od devetih razvijalcev [9]. Produktni vodja in skrbnik metodologije se prištevata v kvoto samo v primeru, ko dejansko opravljata delo razvijalca. Vsi člani ekipe imajo naziv razvijalec, metodologija drugih nazivov ne priznava.

### 4.2.3 Skrbnik metodologije

Skrbnik metodologije je odgovoren za razumevanje in izvajanje same metodologije. Skrbi, da ekipa razume pomen jasnega in natančnega seznama zahtev in da se tega v praksi drži. Produktnemu vodji pomaga iskati načine za uspešno obvladovanje

seznama zahtev. Celotni ekipi pomaga pri razumevanju in uporabi Scrum-a ter organizira dogodke, ki jih predvideva metodologija. Ena ključnih nalog pa je tudi, da skrbi za razumevanje procesa tudi izven ekipe in s tem varuje ekipo pred zunanjimi vplivi, ki bi negativno vplivali na produktivnost [5].

## 4.3 Kritike

Kritike [12] se nanašajo na pretirano zavračanje menedžmenta in poveličevanje samoorganiziranih ekip ter ignoriranje konteksta, v katerem metodologija deluje. Propagira se univerzalna uporabnost in goji nekakšen elitizem – če Scrum projekt ne uspe, je bila to napaka izvajalcev, ki niso razumeli koncepta oz. se niso dovolj potrudili. Na to se navezuje tudi pojem *ScrumBut*, torej ,,Scrum, ampak" – uporabljamo scrum, *ampak* ker . . . , delamo raje . . . Pojem se praktično vedno uporablja v negativnem kontekstu, kar (kljub deklarirani agilni naravi metodologije) nakazuje na dogmatičen pristop nekaterih bolj zagretih navdušencev.

# Poglavje 5

# Prehod in prilagoditve

## 5.1 Prehod

Razvojna ekipa, ki se odloči za agilne metodologije lahko kaj hitro naleti na odpor vodstva ali naročnika. Tak prehod namreč ne vpliva samo na razvojno ekipo, pač pa na vse vpletene v projekt [13]. Odpor je posledica dejstva, da:

- so potrebne spremembe na celotnem projektu, ne le v razvojni ekipi,
- so agilna načela v nasprotju s klasičnim pristopom,
- je potrebna množica sprememb v kratkem času,
- je končni rezultat pogosto nepredvidljiv.

Prehod, ki je izveden brez prave podpore vseh vpletenih, lahko spodleti kljub temu, da znotraj ekipe funkcionira. V našem primeru smo se težav zavedali tako vodstvo, kot razvojna ekipa, pa tudi s strani naročnika ni bilo težav.

## 5.2 Prilagoditve

Zaradi specifičnih lastnosti projekta smo morali nekatere koncepte prilagoditi. Precenili smo, da je projekt dovolj specifičen, da izvirna metodologija ne bi bila najbolj primerna. Vseeno smo poskušali spremembe uvesti kolikor se je dalo v duhu metodologije.

Scrum predvideva homogeno ekipo z medsebojno zamenljivimi člani. Nihče ne sme biti specializiran za določeno področje, saj mora biti vsak sposoben rešiti katerokoli nalogo. Pri ekipi, ki je že v osnovi razdeljena na programerje in svetovalce je to seveda mogoče le znotraj obeh skupin. Ker so od prej ostale neformalne vloge, kdo rešuje katera področja, smo znotraj ekipe konec vsake iteracije uvedli sestanke, na katerih s programerskega in uporabniškega vidika pregledamo vse zaključene zgodbe in na ta način poskrbimo, da smo vsi vsaj na grobo obveščeni o celem projektu. Ker je razvojna ekipa zadolžena tudi za pomoč uporabnikom, je le približno polovica dneva rezervirana za scrum. S stališča metodologije bi bilo seveda bolje, da bi ekipa delala samo na scrumu, bi pa to seveda pomenilo, da bi moral podporo uporabnikom prevzeti nekdo drug, kar pomeni dodatne ljudi, ki pa jih ni lahko dobiti.

Tudi vloga produktnega vodje je nekoliko spremenjena. V optimalnih pogojih bi bil produktni vodja nekdo s strani naročnika, ki bi dobro poznal vse zahteve in bi bil razvojni ekipi vedno na voljo. V praksi to žal ni izvedljivo, saj ob kompleksnosti projekta ni ljudi, ki bi imeli dovolj pregleda nad projektom in bi bili ves čas na voljo razvojni ekipi. Zato je produktni vodja iz našega podjetja.

Čeprav scrum ne predvideva klasičnega projektnega vodje, je ta vloga ostala. Do neke mere pomaga produktnemu vodji pri zbiranju zahtev, sicer pa skrbi za pogodbe in izdane fakture. Poleg tega skrbi za statusne sestanke z naročnikom. Le-ti so spet ostanek klasičnega pristopa, vendar jih naročnik zahteva, zato so ostali. Scrum sicer predvideva dovolj sestankov, na katerih se naročnika obvesti o napredku, vendar se jih naročnik večinoma ne udeležuje – eden od razlogov je zagotovo dobrih 100 km razdalje med sedežema obeh podjetij.

### 5.2.1 Definicija pripravljenosti

Zaradi starih slabih izkušenj z definicijo zahtev, delno rešenimi zahtevki in pomanjkljivim testiranjem smo veliko pozornosti posvetili zagotavljanju, da se to v prihodnosti ne bi dogajalo. Da se uporabniška zgodba lahko vzame v sprint, mora ustrezati definiciji pripravljenosti (*Definition of ready*). Zgodba je pripravljena, ko je spisana v pravilni obliki, za kar praviloma skrbi ekipa na sestankih za definiranje zahtev. Ta oblika ekipi nudi vse podatke, ki jih potrebuje za razvoj:

A. **Jaz kot ( navedite uporabnika / skupino / funkcijo ali službo)**:
Sporočevalec, ki ga lahko po potrebi kontaktiramo za dodatne informacije.

B. **Želim da se (navedite akcijo ali cilj naloge)**:
Zahtevana funkcionalnost. Ključno je, da je opisana *uporabniška* zahteva, torej vsebinska funkcionalnost in ne tehnična rešitev. Za dani primer (**??**) bi bilo neprimerno napisati ,,dodati polje na tabelo kupcev". Iskanje tehnične rešitve je naloga ekipe in ne sporočevalca.

C. **To potrebujem(o) zaradi (navedite razlog)**:
Razlog za to zahtevo nam pove kontekst, s katerim lažje najdemo dobro tehnično rešitev. Obenem nam pomaga ugotoviti, ali je v točki B res opisana uporabniška zahteva.

D. **Merila s katerimi merimo ali je naloga uspešno končana**:
Seznam vseh zahtevanih učinkov na delovanje sistema. Ta seznam je osnova za testiranje in skladnost z DoD. Po potrebi naj vsebuje tudi negativne teste. S tem se zagotovi, da implementacija ne povzroča neželenih stranskih učinkov.

E. **Ali gre za možno razširitev standardne funkcionalnosti (lokalizacija)**?
Ali gre za funkcionalnost, ki ni specifična za konkretnega naročnika in bi bila uporabna tudi za ostale ekipe.

F. **Pravice uporabnikov, ki bodo to uporabljali**?
Razvojna ekipa praviloma programira in testira z administratorskimi pravicami, zato se je pogosto pozabljalo na nastavitve pravic. V ta namen je potrebno izrecno določiti skupine, katerim mora razvijalec omogočiti dostop do funkcionalnosti.

G. **Priložen model/načrt (primer npr. forme, izpisa, načrt dodelave)**:
Dodaten opis, ki vsebuje pomembne informacije o podatkovnem modelu, procesu in ostale podatke, potrebne za razvoj. Polje izpolni ekipa ob planiranju iteracije in ne sporočevalec. Pri preprostih zahtevah, kjer so tehnične zahteve samoumevne je lahko polje prazno.

H. **Zaključna dokumentacija (uporabniška, interna, tehnična navodila)**:
Potrebna dokumentacija. Preverja se za DoD.

A. **Jaz kot (navedite uporabnika / skupino / funkcijo ali službo)**:
računovodstvo

B. **Želim da se (navedite akcijo ali cilj naloge)**:
omogoči vnos davčne številke kupca

C. **To potrebujem(o) zaradi (navedite razlog)**:
davčne zakonodaje

D. **Merila s katerimi merimo ali je naloga uspešno končana**:
vnos davčne številke na kartici kupca
prikaz davčne številke na seznamu kupcev
možnost iskanja kupca po davčni številki
prikaz davčne številke na izpisu odprtih postavk

E. **Ali gre za možno razširitev standardne funkcionalnosti (lokalizacija)**?
da

F. **Pravice uporabnikov, ki bodo to uporabljali**?
saldakonti kupcev

G. **Priložen model/načrt (primer npr. forme, izpisa, načrt dodelave)**:
doda se polje na tabelo CustTable, field group Identification
ime reporta za IOP je CustStatementExt

H. **Zaključna dokumentacija (uporabniška, interna, tehnična navodila)**:
uporabniška navodila, obvestiti stranko

Slika 5.1: Primer uporabniške zgodbe v ustrezni obliki

### 5.2.2 Definicija končane zahteve

Vsako zgodbo v iteraciji najprej prevzame programer, ki sprogramira potrebno kodo. Programer opravi osnovne teste in preda zahtevo svetovalki, kar pomeni da pokaže vse funkcionalnosti in seznam opravljenih testov. Svetovalka testira funkcionalnost in sestavi dokument z vsemi opravljenimi testi. Minimalno, kar mora testirati so vsa merila iz točke D zgodbe. V primeru, da je test uspešen, spiše še predvideno dokumentacijo. Medtem eden od preostalih programerjev naredi pregled kode (*Code review*), s čimer se preveri, ali koda ustreza vsem standardom. S tem, ko je koda pregledana, so vsa merila testirana (kar je razvidno iz dokumenta) in je bila rešitev predstavljena naročniku, se smatra, da je zgodba zaključena.

## 5.3 Primerjava prej in po

Prehod je izboljšal večino večjih težav, s katerimi smo se ukvarjali pri tradicionalnem pristopu. Sprotno spreminjanje prioritet in zahtevkov ostaja, vendar to sedaj ne predstavlja več ovir, saj je metodologija temu prilagojena. Kolikor se morda sliši protislovno, imamo z agilnim pristopom delo bolj urejeno kot prej. Čeprav je vrstni red zahtev natančno določen samo dva tedna vnaprej, se od začetka iteracije tega reda tudi držimo. Pred prehodom je bilo jasno, da so spremembe potrebne, kljub temu, da jih metodologija ne predvideva. Brez pravega načina spopadnja s spremembami pa tudi ni bilo pravega reda.

Z definiranjem natančnih pravil, kdaj je zahteva pripravljena, smo v veliki meri rešili problem slabo definiranih zahtev. Ve se, da zahteva, ki ni definirana v pravi obliki ne more priti v sprint oz. jo je potrebno pravilno definirati, če se mora nujno uvrstiti v naslednji sprint. Občasno se sicer še zgodi, da se med sestankom za definiranje zahtev in začetkom sprinta na vrhu pojavi kakšna nedefinirana zahteva, ki jo je potrebno definirati pred načrtovanjem sprinta, ampak to ne pomeni dodatnega dela, le malo odstopanja od urnika. Če take zahteve ni možno definirati, ne gre v sprint.

Z definiranjem natančnih pravil, kdaj je zahteva končana, smo tudi rešili problem s slabim testiranjem. V zgodbi so jasno zapisana merila, kaj vse zahteva obsega. Da je zgodba končana, morajo biti vsa ta merila testirana, kar mora biti tudi zapisano v dokumentu. Tako se odkrije večino hroščev še v razvoju. Z dobro

definiranim in potrjeno implementiranim obsegom tudi precej bolje obvladujemo naraščanje obsega, saj se ne dogaja, da bi del zahteve ostal do naslednje iteracije, ko bi se mimogrede prikradel zraven še kakšen dodatek, na katerega se je naročnik spomnil šele, ko je začel uporabljati delno implementirano rešitev. Ker so zahteve dobro definirane tudi projektni vodja lažje preceni, ali sodijo v okvir podpisanih pogodb ali gre za plačljive dodelave.

Medtem ko sta vodstvo podjetja in (še posebej) ekipa z veseljem sprejeli zamenjavo metodologije, pa se mi zdi, da naročnik v resnici ni dovolj spremenil načina dela. Posamezni oddelki, od katerih prihajajo uporabniške zgodbe so se sicer privadili na način dela, vseeno pa uprava deluje na enak način, kot prej. Še vedno se načrtuje z delom v fazah, ki predstavljajo vsebinsko zaključeno celoto in trajajo tipično med nekaj meseci in pol leta. Do neke mere je krivda na naši strani, saj bi po mojem mnenju lahko naročniku metodologijo bolje predstavili. Naročnik je sicer prehod na Scrum podprl, vendar menim, da gre bolj za podporo iskanju rešitev, kot pa podporo dejanskim spremembam v (njihovem) načinu dela. Z boljšo predstavitvijo prednosti bi lahko naročniku metodologijo približali do te mere, da bi tudi sami prilagodili način dela.

## 5.4 Potencialne izboljšave

Zelo dobrodošla izboljšava bi bila, če bi imele svetovalke več časa za analizo zahtev. Glede na to, da (vsaj zaenkrat) ni možnosti, da bi bil naročnik ves čas na voljo, bi vsekakor koristilo, če bi zahtevke definirale svetovalke skupaj z naročnikom. Ker vsebinsko dobro poznajo sistem in tudi precej dobro vedo, kaj se potrebuje za razvoj, bi lahko na tak način pravočasno razjasnile podrobnosti, ki jih sedaj sicer ujamemo šele med sestanki (ko naročnik ni prisoten), zaradi česar se izgublja čas. Razloga, da svetovalke tega časa nimajo, sta dva.

Po eni strani ogromno časa porabijo za testiranje, saj testirajo za petimi programerji, kar vzame veliko časa, pogosto pa so testi tudi zelo obsežni. Testiranje sprememb na obračunu storitev ali na knjiženju finančnih dokumentov lahko zahteva množico pozitivnih in negativnih testov, z zamudno pripravo podatkov za vsak test posebej. Ena možnost bi bila, da bi uvedli avtomatsko testiranje [14]. Ker je projekt v osnovi prilagajanje obstoječe kode, ki je ne vzdržujemo mi, je ne-

mogoče pričakovati kakršne koli izčrpne enotske teste. Primeri procesov, ki se jih pogosto testira so npr.: knjiženje fakture, obračun storitev, delovni tok potrjevanja nabavnih nalogov, itd. Za take kompleksne procese je skoraj nemogoče narediti res avtomatiziran test, saj bi bilo potrebno pokriti občutno preveč vhodnih spremenljivk in množico izhodnih zapisov v tabelah. Tudi sama koda ni napisana na način, ki bi omogočal enostavno izolacijo in ločeno testiranje posameznih komponent. Seveda pa vse to ne pomeni, da ni možno avtomatizirati vsaj dela procesa. Velik del testiranja procesov je priprava vhodnih podatkov. Za splošen test obračuna storitev je potrebno pripraviti kakih deset do petnajst primerov vhodnih podatkov, ki pokrije vse osnovne tipe obračuna (akontacija, poračun, konsolidacija, razne akcije...), kar traja približno šest ur. To je sicer najslabši primer, ki ga ni potrebno testirati vsak sprint, vseeno pa bi avtomatizacija priprave podatkov v tem primeru prihranila večji del dneva.

Veliko časa svetovalke porabijo tudi za pomoč strankam. V duhu agilnosti bi lahko v ta namen predlagal kanban, saj je pri kanbanu prioriteta ravno čim bolj sprotno reševanje zahtev. V resnici pa je verjetno edina prava rešitev ločena ekipa za podporo. To je sicer normalna praksa, da se končani implementacijski projekti prenesejo na posebno ekipo, ki se ukvarja samo s podporo in vzdrževanjem starih projektov. Težava je v tem, da je naš projekt tudi znotraj ERP projektov preveč specifičen, da bi ga lahko prevzela podporna ekipa. Če bi torej vseeno hoteli posebno ekipo, bi morali bodisi razbiti trenutno ekipo (kar je neizvedljivo, ker bi v implementacijski ekipi ostalo premalo ljudi), bodisi najeti dodatne ljudi (kar poskušamo že za obstoječo ekipo).

# Poglavje 6

# Sklepne ugotovitve

V pričujočem diplomskem delu sem poskušal čim bolje predstaviti proces, skozi katerega smo na projektu s tradicionalnih prešli na agilne metodologije. Opisal sem težave, s katerimi smo se spopadali in razloge zaradi katerih smo se odločili za prehod. Nekatere težave so se z novo metodologijo rešile, pri drugih vzroki ostajajo, vendar zaradi novih metod ne predstavljajo več težav. Nekatere težave kljub vsemu ostajajo in mislim da v tem naš projekt ni izjema. Razlika je predvsem v tem, da se z današnjimi težavami pol leta nazaj sploh nismo imeli časa ukvarjati. Za nekatere od njih, kot je npr. v prejšnjem poglavju omenjeno testiranje, je jasno v kateri smeri leži rešitev. Za druge, kot je obvladovanje podpore uporabnikom, pa bo potrebno več analize. Smo pa z boljšim testiranjem in posledično manj zahtevami, ki se nanašajo na hrošče, že nekoliko omejili količino podpore.

Izboljšalo se je tudi vzdušje na delovnem mestu. Zaradi konstantnih sprememb, ki jim nismo uspeli slediti, je ves čas vladal pritisk in občutek časovne stiske brez jasnega roka, ki ga lovimo. Sedaj je vzdušje precej bolj sproščeno. Po moji oceni sicer razvoj traja dlje časa kot prej, vendar se rešene zahteve veliko redkeje vrnejo nazaj zaradi napak.

Opisal sem tudi prilagoditve Scruma, ki smo jih uvedli in verjamem, da bi jih marsikateri pristaš Scruma močno skritiziral [15]. Opažam, da je med certificiranimi Scrum strokovnjaki ta kritika zelo pogosta. Trdijo, da se Scruma ne prilagaja in da so pravila z razlogom takšna, kot so ter da ostali spreminjamo Scrum samo zato, ker ga ne razumemo.

V idealnih pogojih bi lahko Scrum uspešno implementirali povsem po pravi-

lih. Po drugi strani bi v idealnih pogojih lahko uspešno uporabili tudi kaskadno metodo. Nekatere okoliščine projekta so enostavno drugačne od tistih, ki jih predvideva Scrum. Seveda ni dobro, da ista ekipa razvija po Scrumu in istočasno mimo tega izvaja še podporo uporabnikom, vendar je lažje modrovati, kot zaposliti nekaj novih ljudi in jih tudi plačevati.

Implementirali smo rešitev, ki nam predstavlja očiten napredek. Rešitev bomo še naprej izboljševali in se morda s tem bolj približali Scrumu. Menim pa, da je razumna izbira prilagoditev boljša pot do cilja, kot strogo sledenje pravilom.

# Literatura


[1] F. Solina, *Projektno vodenje razvoja programske opreme.* Fakulteta za računalništvo in informatiko, 1997.

[2] M. Zorman, "Introducing scrum in the company adacta," *Glasilo PMI Slovenija*, vol. 4, no. 10, 2013.

[3] E. Robinson, "Why crunch mode doesn't work: six lessons," 2012.

[4] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[5] J. Urevc and V. Mahnič, "Ocena prednosti metode scrum in njenih tipičnih praks," *Uporabna informatika*, vol. XX, no. 3, 2012.

[6] D. F. Rico, "What is the ROI of agile vs. traditional methods?," *TickIT International*, vol. 10, no. 4, pp. 9–18, 2008.

[7] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change.* Pearson Education, 2004.

[8] M. Hammarberg and J. Sunden, *Kanban in Action.* Manning Publications Company, 2013.

[9] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum. org, October*, 2011.

[10] K. H. Pries and J. M. Quigley, *Scrum Project Management.* Taylor & Francis, 2010.

[11] W. Grant, "Sprint retrospective techniques." `http://waynedgrant.wordpress.com/2012/04/01/sprint-retrospective-techniques/`, 2012.

[12] P. Krutchen, "Agile's teenage crisis?." http://www.infoq.com/articles/agile-teenage-crisis, 2011.

[13] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st ed., 2009.

[14] B. Muhič, "Automation of functional testing in software development process," Sept. 2014.

[15] M. Vizdos, "Modifying Scrum - You THINK you know better...." http://www.implementingscrum.com/2012/01/18/modifying-scrum-you-think-you-know-better/.